Research Paper | Computer Science | E-ISSN No : 2454-9916 | Volume : 3 | Issue : 5 | May 2017

CODE EVALUATION AND SUGGESTION SYSTEM

Priyanka Sharma [1] | Samkitkumar Jain [1] | Prachin Ranawat [1] | Husain Poonawala [1] | Pankaja Alappanavar [2]

[1] Student, Department of Information Technology, Sinhgad Academy of Engineering, Pune, India.

[2] Professor, Department of Information Technology, Sinhgad Academy of Engineering, Pune, India.

ABSTRACT

Today, due to a large number of online courses there is a need for automatic evaluation of computer programs. Grading of computer programs is also helpful for the companies in their recruitment process where a set of programming statements are to be solved by the students.

Thus, automatic grading of open-ended responses has increasingly become a subject of research. It provides scalability and efficiency. Existing systems which are used for such assessments provide a score on the basis of number of test cases which are passed by the computer program. This does not reflect the overall abilities of the programmer. Blending the existing systems with machine learning and considering the programming styles can prove to be more useful for the evaluation of computer programs.

Thus, our system, which is capable of addressing the above-mentioned issues, has been described in the paper. The paper focuses on how our "Code Evaluation and Suggestion System" can prove to more beneficial for the purpose of automatically evaluating computer programs. It concentrates on using machine learning algorithms combined with the traditional test case system and considering the various programming styles for grading the codes. It is also capable of providing suggestions to the students in order to improve their scores.

KEYWORDS: Program, Random Forest, Machine Learning, Natural Language Processing.

I. INTRODUCTION

These days, a number of people are enrolling for online programming courses, where they have to solve computer programming questions. Such questions are also often used by various companies in their recruitment process to test and hire candidates. These questions consist of problem statements wherein the students are required to submit their solution in the form of a code written in the programming language specified. The solutions are then checked with the test cases specific to each problem statement and the result is generated based on the number of test cases passed. Evaluation of a code in such manner requires more time and can lead to rejection of some capable candidates. Hence, here arises a need to grade such computer programs according to the efficiency of the codes and not only with the test cases. It could be used to reduce the dependency on the availability of manual evaluators and also help improve self-assessment.

This paper mentions a machine learning solution, using the Random Forest Algorithm, which is capable of scoring the efficiency of the codes. The paper has been organized into seven sections as follows. Related research in this area is presented in Section 2. Section 3 gives a brief outline of the Random Forest algorithm. A description of the features given as input to the ML algorithm is mentioned in Section 4. The methodology used has been described in Section 5. Section 6 provides the analysis of the results. Conclusion and our expectations related to the future trends in automatic assessment of programming assignments are discussed in Section 7.

II. RELATED WORK

A number of approaches have been made to evaluate computer programs. Many surveys have also been done in this field due to its increasing popularity [1].

Machine learning is one such approach which is used in Automatic Grading of Computer Programs: Machine Learning Approach by Shashank Srikant and Varun Aggarwal [2]. This machine learning approach derives a set of features such as presence of basic keywords or control structures from a given program to evaluate the program code. These features are then used to learn a machine learning model to grade the programs on the basis of certain rules.

Many researchers have used styling as a factor to impart scores. The ability to write nice and elegant programs is a very important skill which is not usually the focus of programming courses [3]. A programming style is considered to be an individual's interpretation of rules and their application to the writing of source code in order to achieve the aim [4]. Automatic analysis of functional program style by Greg Michaelson [3] describes semantic style rules and has also presented an automatic style analyser. Thus, paying more attention to the style of programming aids the evaluation of programs.

For the purpose of evaluating computer program codes, a number of universities use a methodology of peer-assessment where the submitted programs are evaluated by other students belonging to the same class. This helps in situations where the manual evaluators are not available and also allows students to understand their mistakes while evaluating the codes. This method can be combined with the web technology and has been stated by Jirarat Sitthiworachart and Mike Joy [5].

III. MACHINE LEARNING: RANDOM FOREST

A machine learning (ML) model is constructed by training a ML algorithm which uses the training data (input data) for its construction. And later on makes use of the testing data to check the correctness of the generated results.

The algorithm used in this paper is the Random Forest algorithm. We chose Random forest for two reasons. First, it has been successfully used for regression problems. Second, Random Forest is an effective learning algorithm.

The algorithm is based on decision trees working in an ensemble. It is developed from two successful approaches [6]. The first one is an ensemble of trees where each tree grows from the training set via bagging. This reduces the generalization error which is caused due to combining decisions of multiple models which are usually weak and unstable individually. The second approach is random split selection for a decision tree. This split is chosen randomly from a subset of best splits.

Each tree is grown using the obtained bootstrap sample. The second mechanism involves randomly selecting a small fraction of features and further splitting using the best feature from this set. The size of a fraction (i.e. the number of features to select) is fixed within the algorithm execution.

It has only two main parameters affecting its performance considerably: The number of trees, m in an ensemble to grow and the number of features, k to select randomly at each split.

IV. FEATURES

For the purpose of providing a more efficient way for the evaluation of codes, the below mentioned features are considered. These are problem independent features.

- **Test cases:**

The code gets tested against some basic test cases.

A low test case score indicates that the student has to enhance his ability of logical thinking and his/her in-depth understanding of the question.

- **Execution time:**

This represents the time required for the execution of the code. A lower execution time indicates the ability to code efficiently by keeping in mind the time complexity and a higher value indicates some work is needed for the selection of data structures, algorithms to reduce the time complexity.

❖ Space :
This represents the space/memory resources required by the code. It indicates the ability to code efficiently by keeping in mind the space complexity. Higher value represents a good selection of data structures, variable data types to reduce the space complexity.

❖ **Error:**
The number of errors/bugs occurred during the code compilation are shown by this feature.

❖ **Warning:**
This represents the number of warnings occurred during the compilation of the code and provides suggestions to prevent bugs.

❖ **Style:**
It represents the stylistic issues related to the submitted code like unused functions, redundant code.

❖ **Performance:**
It provides suggestions for making the code faster. These suggestions are only based on common knowledge. It is not certain that any measurable difference in speed can be achieved by fixing these messages.

❖ **Portability:**
It gives portability warnings. The score is based on the portability of code for future similar problems. 64-bit portability code might work different on different compilers.

❖ **Information:**
It provides information about any configuration problems.

❖ **Build:**
This feature shows if any deprecated functions are present in the code. It is needed to check if the functions being used are compatible with the current version.

❖ **Legal:**
It provides the legal information about the code.

❖ **Readability:**
It is used to check whether the code is readable or not. This feature has great importance from the maintenance point of view of codes.

❖ **Runtime:**
This feature provides the time for executing the code submitted by the candidate taking the test.

❖ **Whitespace:**
It is used to give the amount of white space/blank space used in the code by the candidate.

There can be other features also which can prove to be helpful for the evaluation of codes but the above mentioned features are more likely to grade the programs with a higher degree of accuracy. These are the primary features which a manual evaluator would look at while evaluating the codes. They focus on the most important parameters of the code such as time complexity, space complexity, and redundancy. Also they are independent of the problem statement provided and can be easily extracted. This was our main purpose for the selection of these features in particular.

V. ARCHITECTURE

The various components of the system have been mentioned in the figure below. It also shows how these components interact with each other and transfer the processed data. The entire system architecture is divided into 5 modules.

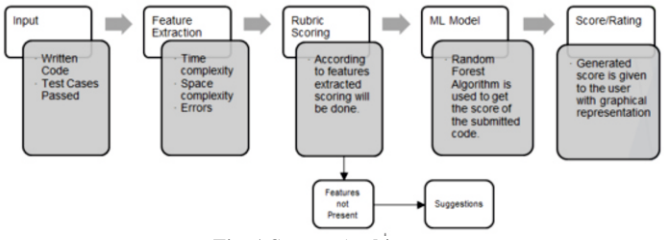

Fig. 1 System Architecture

1. Input:
This module accepts the code submitted by the user and checks it against the test cases to get the number of passes test cases. This number is then given to the next module along with the code.

2. Feature Extraction:
Features, mentioned in Section 4, are extracted from the accepted code from the user. Our primary motivation to address the problem of automatic evaluation is to try and understand the aspects, which a manual evaluator would consider when grading a program.

3. Rubric Scoring:
All the extracted features are given individual scores. This is used as the training data for the next module of machine learning. If certain features are not present in the submitted code then the user is given suggestions on how he/she can work upon them to increase their scores.

4. ML Model :
A machine learning model using Random Forest Algorithm from the above feature scores is created and trained. And further this model is used to predict score of newly submitted codes.

5. Score / Rating:
Lastly, a final score is calculated for the submitted code using the scores gained by machine learning model and aggregating them.

Sample code:

```
#include <stdio.h>
int main(void) {
// your code goes here
int i,j;
for(i=0;i<5;i++)
{
  for(j=0;j<i;j++)
  {
     printf("%d",i);
  }
  printf("\n");
}
return 0;
}
```

Test cases: 9

Output:

{"message":"Success","features":{"testcases":"9","warning":0,"readability":0,"information":0,"style":0,"whitespace":44,"portability":0,"space":2567,"extime":0.04,"legal":1,"build":0,"error":0,"y":3.82376769369,"performance":0,"runtime":0}}

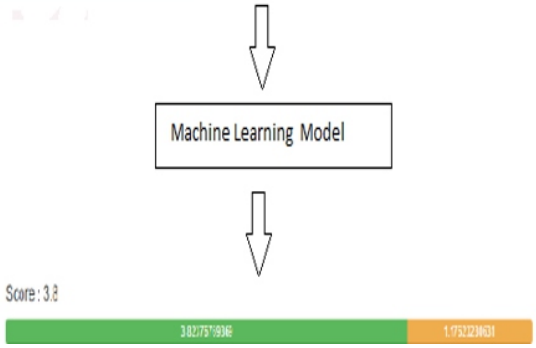

Fig. 2 Input/output for a sample code.

As show in the figure, feature scores are obtained from compiling the sample code and are then given to the ML model which gives the final score of the sample code as 3.8 out of 5 by using the random forest algorithm.

VI. EXPERIMENTATION

For generating the grades automatically for various computer programs, following method was done.

A. Dataset
The initial dataset comprised of 1000 codes submitted by engineering undergraduate students. The collection of the dataset was done by conducting a coding event where students submitted their solutions to the various computer programming problems in both C and C++ languages. Once the codes were submitted, feature extraction was done for the features discussed in Section 4 for creating the dataset for machine learning purpose.

The dataset comprised of the code no (given manually) and the 13 features. For the purpose of learning our model, 70% of the dataset was taken as the training data and 30% as the testing data to check for the correctness of our model results.

B. Modelling

The training data (obtained from the above step) was given as input to the random forest algorithm for the learning purpose. Once the ML model was trained with the features and the grades, the test data was then fed as input to the model.

The output obtained was the grade of the student for a particular problem based on the 13 features. Being an ensemble method, the random forest algorithm generates multiple learners which give different output for the same problem. In order to obtain a single output value as the grade, an average of the values from each learner was taken by the algorithm.

The final score given was out of 5.

C. Observations

We observed that by increasing the number of trees in the random forest algorithm for the construction of the machine learning model, the mean square error reduced.

VII. RESULT ANALYSIS

After the results were generated, it was seen that a student, who would have been otherwise rejected by the existing evaluation system, was selected when evaluated by our mentioned system. The reason is largely due to the other features present in a code which were also extracted. This is represented graphically below.

Fig.3 depicts the scores of the candidates generated when the existing system is used compared with our system.i.e.; using features along with the test cases. Thus, this shows a significant decrease in the number of rejected candidates as compared to the previously existing systems.

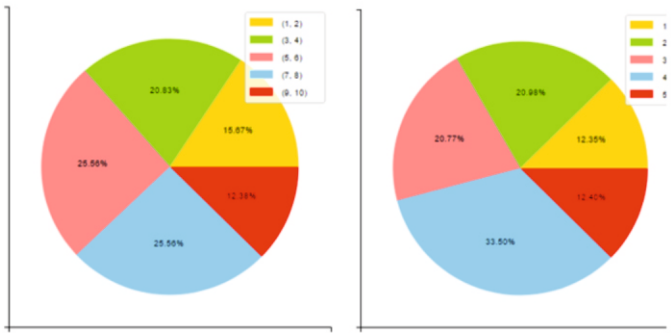

Fig.3 System with testcases vs our system

Also, we made a comparison for SVR (Support Vector Regression) with Random Forest algorithms and found out that the error in random forest is lesser than that of the SVR. Random Forest has a lesser mean square error as compared to the SVR algorithm.

VIII. CONCLUSION

This paper helps to study and understand the different features which can be used for automatic evaluation of computer programs, apart from the traditional test case system. The paper discusses those various features and how they can be used in collaboration with machine learning for the assessment of computer program codes. It also mentions how random forest can be used in such automatic evaluations as it provides the result by averaging. Hence reducing the chances of errors and over fitting which is a major concern of decision trees.

IX. FUTURE SCOPE

The study on code evaluation can be further extended by using other machine learning algorithms and using codes in other languages such as Java, Python.

Also different code related features can also be tested for more efficient grading of computer codes such as the number of the lines of code, presence or absence of comments.

ACKNOWLEDGEMENT

We take this opportunity to thank all the people involved in the making of this paper. We want to especially thank our respected guide, Ms. Pankaja Alappanavar for her continuous support and guidance, which motivated us to achieve our goal. Her valuable advice is always appreciated. We also convey our gratitude to our external guide, Mr Dinesh Kulkarni, for encouraging us and helping us whenever needed. We would also like to express our gratitude towards the Head of Department, Prof. Abhay Adapanawar. Lastly, we would like thank our families, the entire teaching and non-teaching staff of our department, and our friends for their much needed valuable suggestions and support.

REFERENCES

1. Priyanka Sharma, et al. (February 2017) .A Literature Survey on Automatic Evaluation of Computer Programs. International Journal of Advanced Research in Computer and Communication Engineering, Vol. 6, Issue.
2. Shashank Srikant & Varun Aggarwal. (2013). Automatic Grading of Computer Programs: A Machine Learning Approach.12th International Conference on Machine Learning and Applications.
3. G. Michaelson. Automatic Analysis of functional program style.
4. Mohan and N.Gold. (2004). Programming style changes in evolving source code.
5. Sitthi Worachart, J., and Joy, M. (ICALT'04). Web-based Peer Assessment System with an Anonymous Communication Tool. Proceedings of the IEEE International Conference on Advanced Learning Technologies.
6. Viachaslau Sazonau. Implementation and Evaluation of a Random Forest Machine Learning Algorithm. University of Manchester, UK.
7. Rathod, Aakash, Nitika Sinha, and Mrs Pankaja Alappanavar. (2016). "Extraction of Agricultural Elements using Unsupervised Learning." Imperial Journal of Interdisciplinary Research 2.6.